типичных эталонов, покрывающих по характеристикам качества шкалу оценок. (Программное обеспечение тестовой системы должно предусматривать сбор необходимой статистики для формирования как эталонной тестовой выборки для обучения НС, так и получение соответствующих данных, используемых НС на этапе экспертного оценивания качества тестовой выборки в целом. Например, такими данными могут являться относительные частоты событий, характеризующих, по мнению исследователя, качество тестового задания. Выходные эталоны содержат значения, соответствующие определенному рангу качества эталона).

3) Выполнить обучение нейронной сети до достижения приемлемой погрешности.

В процессе обучения реализуются обобщающие свойства нейронной сети, на основании чего модель способна пролонгировать результаты обучения и в процессе функционирования выполнить задачу эксперта: путем сканирования имеющихся тестовых наборов выдать оценки качества тестовых заданий.

Заключение. Предлагаемый в работе подход к организации модели оценки тестовых выборок имеет следующие особенности: дает возможность динамически по мере необходимости изменять обучающую выборку оценок, а, следовательно – адаптивно изменять свойства функции оценки; позволяет избежать формализации модели оценивания; инвариантен относительно критериев оценивания. В данном контексте подход может быть более широко применен в задачах оценки качества тестирования программного обеспечения различного назначения.

Вместе с тем нейросетевые модели обладают повышенной временной сложностью процесса обучения, высокими требованиями к репрезентативности обучающего множества, что обуславливает необходимость наличия определенных навыков в использовании НС при решении практических задач подобного класса [5, 6].

Работа выполняется в рамках НИР ЭИ-08/06 «Современные интеллектуальные технологии обработки информации», ГР № 2008553.

СПИСОК ЦИТИРОВАННЫХ ИСТОЧНИКОВ

1. Моисеев, В.Б. Оценивание результатов тестирования на основе экспертно-аналитических методов / В.Б. Моисеев, В.В. Усманов, К.Р. Таранцева, Л.Г. Пятирублевый // «Открытое образование», №3. – 2001. – С. 32–35.
2. Hertz J., Krogh A., Palmer R. Introduction to the Theory of Neural Computation. – Addison Wesley Publishing Company. – 1991. – 327 p.
3. Kroese B. An Introduction to Neural Networks. – Amsterdam: University of Amsterdam. – 1996. – 120 p.
4. Оссовский, С. Нейронные сети для обработки информации / Пер. с польского И.Д. Рудинского. – М.: Финансы и статистика, 2002. – 334 с.
5. V. Golovko, Y. Savitsky, N. Maniakov. Neural Networks for Signal Processing in Measurement Analysis and Industrial Applications: the Case of Chaotic Signal Processing // chapter of NATO book “Neural networks for instrumentation, measurement and related industrial applications”. - Amsterdam: IOS Press, 2003, pp. 119-143.
6. V. Golovko, Yu.Savitsky, Th.Laopoulos, A.Sachenko, L.Grandinetti. Technique of Learning Rate Estimation for Efficient Training of MLP // Proc. of Int. Joint Conf. on Neural Networks IJCNN’2000, Como, Italy. Vol. 1. – 2000. – P. 323–329.

Материал поступил в редакцию 25.10.09

SAVITSKY Y.V. A Neural Network Based Technique for Modeling of a Quality Testing Set

A Technique for adaptive modeling of a quality testing set using an artificial neural network is discussed. A neural network arrangement is grounded and formulated; a methodic for training set organization is proposed; a common algorithm for the model building is described. Perspectives of application of this approach to the more wide set of tasks are presented.

УДК 004.272.26

Уваров А.А., Садыхов Р.Х.

ВЫЧИСЛИТЕЛЬНАЯ КОНФИГУРАЦИЯ ДЛЯ РЕАЛИЗАЦИИ НА АРХИТЕКТУРЕ CUDA АЛГОРИТМА ФИЛЬТРАЦИИ ИЗОБРАЖЕНИЙ

Введение. За последние годы сформировалось новое направление в параллельных технологиях программирования: использование процессоров графических адаптеров (GPU) для решения универсальных вычислительных задач. Это направление получило название – GP GPU (General-purpose graphics processing units) [1]. Использование вычислительной мощности графических процессоров нашло применение в различных отраслях высокопроизводительных вычислений: моделирование климатических процессов, биомедицина, нефтегазовая отрасль, анализ инженерных конструкций, финансовый анализ, обработка видео, изображений и сигналов [1]. Графические процессоры последнего поколения обладают пиковой производительностью в 1000GFLOPS и в некоторых реальных задачах в 10–100 раз превосходят по производительности универсальные процессоры.

Алгоритм фильтрации изображения [2] хорошо подходит для реализации на массивно-параллельной архитектуре GPU, так как обладает высокой степенью параллелизма на уровне данных, что обеспечивается блочным характером обработки. Обрабатываемые блоки, как правило, состоят из обрабатываемого пикселя и некоторой его окрестности. Причем значение пикселя изменяется в диапазоне от 0 до 255, поэтому для его хранения достаточно 8 бит.

Архитектура CUDA (Compute Unified Device Architecture) [3] разработана компанией NVIDIA для программирования GPU собственного производства. Эта архитектура относится к типу архитектур с массовым параллелизмом. Обработка данных выполняется множеством тредов, которые используют одну и ту же вычислительную функцию, которая называется ядром. Треды группируются в блоки тредов. которые формируют решетку блоков. Исполнительная конфигурация определяет геометрию блока тредов и геометрию решетки блоков тредов. Под геометрией объекта здесь понимается его размерность и размер. Каждый блок тредов выполняет обработку своего блока данных. Вычислительная конфигурация включает в себя исполнительную конфигурацию, геометрию блоков данных, а также способ распределения данных между тредами.

Основная задача, которая ставится в статье – найти оптимальную вычислительную конфигурацию для алгоритмов двухмерной фильтрации изображения с различными размерами масок. Вычислительная конфигурация считается оптимальной, если позволяет достичь максимального быстродействия.

Архитектура CUDA и микроархитектура GPU. Архитектура CUDA налагает некоторые ограничения на исполнительную конфигурацию. Размерность блоков тредов и решетки блоков может принимать значения из множества 1, 2, 3. Количество тредов в одном блоке для текущей архитектуры ограничено 512, а количество блоков в решетке блоков ограничивается по каждой из размерностей по 65535. Исполнительная конфигурация задается в момент запуска ядра и не может быть изменена динамически.

***Уваров А.А.**, аспирант кафедры электронно-вычислительных машин Белорусского государственного университета информатики и радиоэлектроники.*
Беларусь, БГУиР, 220013, г. Минск, ул. П. Бровки, 6.

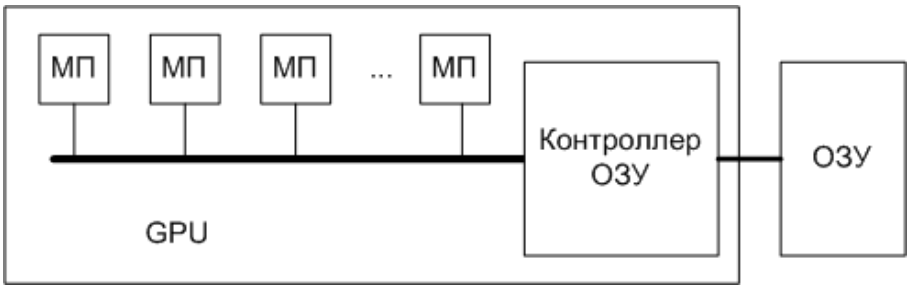

Рис. 1. Схема графического процессора

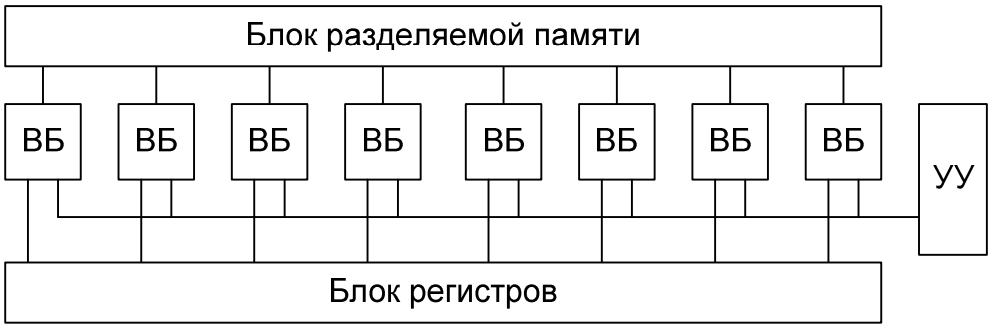

Рис. 2. Устройство мультипроцессора

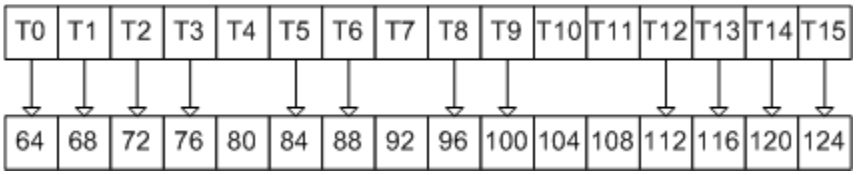

Рис. 3. Пример формирования транзакции

На рисунке 1 приведена общая схема устройства графического процессора. Основные вычисления выполняются массивом мультипроцессоров (МП), их количество может достигать 30. Все мультипроцессоры подключены к контроллеру глобальной памяти. Контроллер памяти находится на кристалле, а глобальная память – вне кристалла видеопроцессора.

На рисунке 2 приведена схема устройства одного мультипроцессора. Мультипроцессор состоит из: 8 одинаковых вычислительных блоков (ВБ) для вещественной арифметики и простых целочисленных инструкций, блока разделяемой памяти, блока регистров, одного устройства управления (УУ).

Мультипроцессор использует парадигму SIMT(single instruction multiple thread). Эта парадигма похожа на парадигму SIMD (single instruction multiple data), но имеет некоторые отличия. В случае SIMT процессор одновременно выполняет большое количество тредов, каждый тред при этом обрабатывает один элемент данных. В случае SIMD выполняется один поток инструкций, который обрабатывает вектор данных. В отличие от парадигмы MIMD(multiply instruction multiple data), треды на SIMT процессоре исполняют один код и выполняются синхронно. Синхронное исполнение обозначает, что в один момент времени все треды исполняют одну инструкцию. Это позволяет упростить процессор за счет использования только одного устройства управления для всех тредов. Варп (warp) – это группа тредов, для которых поддерживается синхронное исполнение. Для современных устройств варп содержит 32 треда. Блок тредов обрабатывается одним мультипроцессором, для исполнения он разбивается на варпы. Между тредами, которые принадлежат различным варпам, автоматическая синхронизация не поддерживается, но мультипроцессор содержит специальную инструкцию для выполнения барьерной синхронизации всех тредов в блоке.

Блок разделяемой памяти принадлежит блоку тредов. Это значит, что любой тред блока может обратиться к любой ячейке памяти. За целостность общей памяти отвечает программист. Физически все треды, которые исполняются на одном мультипроцессоре, используют общий блок регистров, но при этом тред не может напрямую обратиться к регистрам другого треда.

Подсистема памяти GPU, с точки зрения программиста, значительно отличается от таковой у универсального процессора. В архитектуре CUDA существует два основных типа памяти: глобальная и разделяемая. Глобальная память расположена на плате GPU вне кристалла, характеризуется большим объемом и высокой латентностью. От момента поступления запроса до выдачи данных проходят сотни тактов. Разделяемая память, расположенная на кристалле у каждого мультипроцессора, обладает латентностью в несколько тактов. По сути, локальная память является аналогом КЭШа универсального процессора, только за ее управление отвечает программист.

При работе с памятью нужно придерживаться двух основных правил. Во-первых, необходимо максимально сократить количество обращений к глобальной памяти и использовать разделяемую память мультипроцессора как кэш для данных всех тредов блока. Во-вторых, каждое обращение к глобальной памяти должно выполняться максимально эффективно. Для этого отдельные обращения должны объединяться в транзакции.

Все обращения к памяти в архитектуре CUDA обрабатываются одновременно для половины варпа, то есть для 16 тредов. На рисунке 3 приведен пример обращения к памяти, которое приводит к формированию транзакции. Длина формируемой транзакции составляет 64, 128 или 256 байт. Для формирования такого объема данных полуварпом каждая инструкция должна обрабатывать 32, 64, 128 бит соответственно. Блок транзакции должен быть выровнен в памяти на границу своего размера. При этом первый тред полуварпа должен всегда обращаться к первому элементу блока, второй – ко второму и т.д. Порядок обращения для всех тредов должен сохраняться. Допускается, что некоторые треды не будут выполнять обращение к памяти.

Один мультипроцессор может обрабатывать несколько блоков тредов параллельно. Максимальное количество тредов, обрабатываемых одним мультипроцессором со всех блоков, равно 768. При

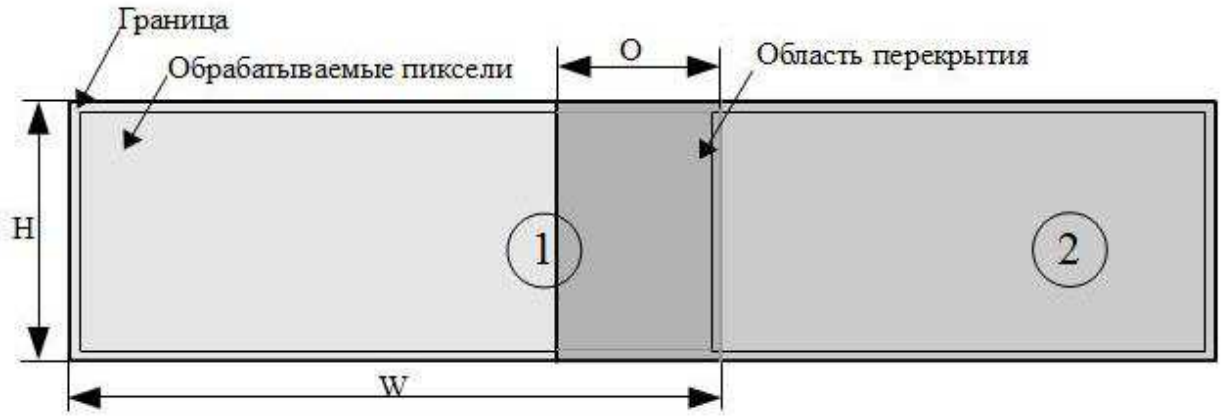

Рис. 4. Схема вычислительной конфигурации

этом общее количество требуемых всеми тредами ресурсов процессора не должно превосходить их реальное количество. Основными ресурсами процессора являются регистры и разделяемая память. При программировании ядер и задании исполнительной конфигурации необходимо стремиться исполнять на одном мультипроцессоре как можно больше тредов. В отличие от универсального процессора, который использует внеочередное исполнения для того, чтобы скрыть латентность инструкции, графический процессор использует переключение варпов для этой же цели. Если данные для текущей инструкции в варпе еще не готовы, то вместо ожидания готовности данных графический процессор переключится на исполнение другого варпа. Переключение между варпами занимает всего один такт. Для того чтобы скрыть латентность типичной арифметической операции, необходимо, чтобы на мультипроцессоре исполнялось 6 варпов или 192 трэда.

Описание вычислительной конфигурации алгоритма фильтрации изображения. Для обработки изображение используется двумерная вычислительная конфигурация. Каждому блоку данных для обработки ставится в соответствие один блок тредов.

Реализация алгоритма фильтрации состоит из трех последовательных действий: чтение данных из глобальной памяти в разделяемую, выполнение фильтрации, запись результата в глобальную память. Для выполнения фильтрация требуется пиксель и его окрестность. Поэтому на первом этапе из глобальной памяти читаются обрабатываемые пиксели и их окрестности. Размер читаемого блока из глобальной памяти больше размера обрабатываемого блока на размер необрабатываемой границы. Всем тредам в блоке в соответствие ставится одинаковое количество пикселей из прочитанного блока. Те треды, которым в соответствие поставлены пиксели границы, выполняют только первый этап, на втором и третьем этапе они простаивают.

Первый и третий этапы алгоритма выполняют только операции обращения к памяти, поэтому время их исполнения зависит от пропускной способности памяти. Для того чтобы добиться максимальной пропускной способности, как сказано в предыдущем разделе, необходимо чтобы обращения к памяти формировали транзакции.

На рисунке 4 представлена схема вычислительной конфигурации, которая удовлетворяет данному условию. Символами 1 и 2 обозначены два блока данных, принадлежащих одной строке, при этом блок под номером 1 является первым блоком строки. Символом W обозначена ширина блока, символом H – высота блока, символом O – величина перекрытия двух блоков. Размер рабочей области первого блока равен $(W\text{-}2*R)*(H\text{-}2*R)$, где R – радиус фильтра Размер рабочей области для второго и последующих блоков равен $(W\text{-}O)*(H\text{-}2*R)$. В рабочей области находятся только обрабатываемые пиксели. Начало каждого блока выровнено на границу 64 байта. Выравнивание первого блока в строке обеспечивается приведением ширины изображения к ближайшему большему числу, кратному 64. Второй и все последующие блоки строки выравниваются сдвигом влево, до ближайшего адреса, кратного 64. Выравнивание начала блока на границу 64 байт необходимо для формирования транзакции. Использование такого подхода для алгоритма фильтрации изображений впервые было описано в работе [4].

Каждый пиксель изображения представлен одним байтом. Для формирования транзакции необходимо, чтобы тред читал 4 байта. Это условие выполняется, если каждый тред будет читать и писать не один, а 4 соседних пикселя. Недостатком этого подхода является то, что ширина горизонтальной границы должна быть кратна 4. Вторым вариантом решения этой проблемы является использование 32-ух бит для хранения одного пикселя. Третьим необходимым условием для формирования транзакций при обращении к памяти является то, что пиксель, расположенный в левом верхнем углу блока, всегда должен читаться нулевым тредом в блоке.

Выбор оптимальных значений параметров W, H, O выполняется по критериям максимизации использования тредов в блоке и минимизации операций чтения из памяти. Коэффициент Km показывает отношение объема выбранных из памяти данных к объему рабочей области. Kt представляет собой отношение количества тредов, которые принимают участие в обработке пикселей рабочей области, к общему количеству тредов в блоке. Коэффициент Kpt показывает, сколько пикселей обрабатывается одним тредом. Размер блока данных не должен превышать объем разделяемой памяти. Если требуется, чтобы один мультипроцессор обрабатывал два блока трэдов, тогда размер блока данных должен быть меньше половины разделяемой памяти. Блок тредов может содержать 256 или 512 тредов.

В процессе анализа функции коэффициента Km было выяснено, что ширина блока, при которой этот коэффициент достигает минимума определяется соотношением

$$w_d = \sqrt{\frac{32 \cdot S_S}{R}} \cdot \frac{1}{sizeof(pixel)}, \qquad (1)$$

где R – радиус фильтра, S_S – объем разделяемой памяти, используемой для хранения блока обрабатываемых данных, в байтах.

На геометрию блоков данных накладываются дополнительные ограничения. Во-первых, ширина блока должна являться степенью двойки. Во-вторых, высота блока данных зависит от высоты блока тредов, который выполняет обработку. Высоту блока тредов можно вычислить по формуле:

$$h_t = \frac{4 \cdot S_{bt}}{w_d}, \qquad (2)$$

где S_{bt} – количество тредов в блоке тредов. Количество строк в рабочей области должно быть кратно высоте блока тредов. Это позволит равномерно распределить работу между всеми тредами. Коэффициент Kpt можно определить по формуле:

$$K_{pt} = \frac{4 \cdot (h_d - 2 \cdot R)}{h_t}. \qquad (3)$$

Таблица 1

Радиус	Разделяемая память	Тредов в блоке	Оптимальный размер	*Km*	Альтернатива 1	*Km*	Альтернатива 2	*Km*
1	8192	256	512x16	1,31	256x30	1,52	512x14	1,52
1	16384	256	724x23	1,2	512x30	1,31	1024x15	1,31
1	16384	512	724x23	1,2	512x30	1,31	1024x14	1,42
2	8192	256	362x23	1,47	256x28	1,77	512x14	1,82
2	16384	256	512x32	1,31	256x60	1,52	512x30	1,41
2	16384	512	512x32	1,31	256x60	1,52	512x28	1,52
3	8192	256	296x28	1,62	256x30	1,77	512x14	2,3
3	16384	256	418x39	1,39	256x62	1,52	512x30	1,52
3	16384	512	418x39	1,39	256x62	1,52	512x30	1,52
4	8192	256	256x32	1,77	256x28	2,13	512x14	3,05
4	16384	256	362x45	1,46	256x60	1,64	512x30	1,66
4	16384	512	362x45	1,46	256x56	1,77	512x28	1,83

Обработка пикселей тредами должна выполняться четверками. Каждый тред последовательно обрабатывает четыре соседних пикселя, а затем переходит к следующей четверке, что позволяет избежать конфликтов при обращении к разделяемой памяти, как описано в [5]. Порядок привязки тредов к четверкам пикселей может быть произвольным.

В таблице 1 приведены оптимальные размеры блоков данных для фильтров с радиусом от 1 до 4. Для каждого фильтра рассматривается три варианта: при объеме блока разделяемой памяти в 8192 байт и 256 тредах в блоке, при объеме блока разделяемой памяти в 16384 байт и 256 тредах в блоке, при объеме блока разделяемой памяти в 16384 байт и 512 тредах в блоке Пиксель представлен типом данных байт. Для каждого оптимального размера блока предложено два альтернативных размера, которые удовлетворяют приведенным выше требованием к геометрии. Для оптимального и для каждого альтернативного размера вычислен коэффициент *Km*.

Размер альтернативных блоков меньше объема разделяемой памяти, так как часть разделяемой памяти используется компилятором для передачи параметров ядру. Этот объем незначителен и составляет около сотни байт. Поэтому размер блока данных был уменьшен по высоте до ближайшего значения, кратного высоте блока тредов.

Предложенные в таблице 1 вычислительные конфигурации необходимо оценить по коэффициенту *Kt*. Чем меньше значение данного коэффициента, тем меньший процент тредов участвует непосредственно в вычислениях. Это особенно актуально для фильтров больших размеров. Например, для фильтра 5x5 объем вычислений больше в 2,7 раза по сравнению с фильтром 3x3. Коэффициент *Kt* вычисляется в соответствии с

$$K_t = \frac{w_d - 64}{w_d}, \qquad (4)$$

где w_d – ширина блока данных. Значение *Kt* для блока зависит только от его ширины, для блоков с шириной 256, 512, 1024 значение *Kt* будет соответственно 0,75; 0,875; 0,9375. На основе значения коэффициента *Kt* можно сделать вывод, что вычислительные конфигурации с шириной блока, равной 256, из дальнейшего рассмотрения можно исключить, так как использование тредов для вычислений у них на 15% меньше чем при ширине блока 512.

Выбор оптимальной вычислительной конфигурации для алгоритма фильтрации изображения. Для того, чтобы выявить оптимальную вычислительную конфигурацию для каждого из фильтров были разработаны соответствующие ядра. Все ядра выполнялись на GPU NVIDIA GeForce 9800GT, который содержал 14 мультипроцессоров, работающих на частоте 1650МГц, пропускная способность памяти 60Гбайт/с, объем памяти на плате 512МБайт. Тестирование проводилось на полутоновом изображении размером 10 миллионов пикселей. Ширина изображения для каждой вычислительной конфигурации определялась шириной блока данных таким образом, что изображение разбивалось на целое число блоков.

Количество используемых регистров для каждого ядра зависело от размера разделяемой памяти и количества тредов в блоке. Ядро использовало 32 регистра, если исполнялось 256 тредами и 16384 байта разделяемой памяти. Во всех остальных случаях ядро использовало 16 регистров. Это ограничение связано с общим количеством регистров, доступных блоку тредов [3].

Таблица 2

Радиус	Разделяемая память	Тредов в блоке	Размер блока	Время мс
1	8192	256	256x28	2,03
1	8192	256	512x14	1,76
1	16384	256	512x30	1,84
1	16384	256	1024x15	1,94
1	16384	512	512x30	1,86
1	16384	512	1024x14	1,99
2	8192	256	512x14	3,92
2	16384	256	512x30	3,79
2	16384	512	512x28	11,35
3	8192	256	512x14	10,59
3	16384	256	512x30	10,36
3	16384	512	512x30	10,63
4	8192	256	512x14	15,68
4	16384	256	512x30	15,52
4	16384	512	512x28	15,92

Данные измерений приведены в таблице 2. На основе анализа данных эксперимента можно выделить две оптимальные конфигурации, которые могут быть использованы для всех размеров фильтров. Это вычислительные конфигурации с количеством тредов в блоке, равным 256, и размерами блоков данных 512x14, 512x30. Вычислительная конфигурация с размером блока 512x30 оптимальна для фильтров с радиусом 2-4, а вычислительная конфигурация с размером блока 512x14 оптимальна для фильтра с радиусом 1. Если делать выбор в пользу единственной конфигурации, то более предпочтительной является конфигурация 512x14, так как она опережает 512x30 для фильтра 3x3 на 4,5% и отстает для фильтра 5x5 на 3,5%.

При анализе таблицы 2 обращает на себя внимание большая разница по скорости между фильтрами 5x5 и 7x7. Это связано не с вычислительной конфигурацией, а с тем, что из-за ограниченного количества регистров компилятор не может использовать технику развертывания циклов.

Заключение. В статье продемонстрирован способ построения вычислительной конфигурации для алгоритмов сверточной фильтрации изображения различного радиуса. Способ позволяет создавать вычислительные конфигурации, которые для всех обращений к глобальной памяти формируют транзакции. Для выбора оптимальных размеров блоков данных были сформированы математические

критерии. На их основе для каждого радиуса фильтра было сформировано множество вычислительных конфигураций. Экспериментально было продемонстрировано, что для фильтров с радиусом 1–4 существует две оптимальные конфигурации с размерами блоков данных 512x14 и 512x30 байт соответственно. В качестве обобщения можно добавить, что найденные вычислительные конфигурации будут оптимальны для алгоритмов обработки изображений, которые используют тот же размер окрестности пикселя, что и рассмотренные алгоритмы фильтрации.

СПИСОК ЦИТИРОВАННЫХ ИСТОЧНИКОВ

1. General-Purpose Computation Using Graphics Hardware [Electronic resource] / 2009. – Mode of access: http://www.gpgpu.org. – Date of access: 10.09.2009.
2. Гонсалес, Р. Цифровая Обработка изображений. / Р. Гонсалес, Р. Вудс – Москва: Техносфера, 2005 – 1072 с.
3. NVIDIA CUDA Programming Guide [Electronic resource] / NVIDIA. – 2009. – Mode of access: http://developer.download.nvidia.com/compute/cuda/2_3/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf . – Date of access: 10.09.2009.
4. Image Convolution with CUDA [Electronic resource] / NVIDIA. – 2009. – Mode of access: http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/convolutionSeparable/doc/convolutionSeparable.pdf. – Date of access: 10.09.2009.
5. NVIDIA CUDA Best Practices Guide. [Electronic resource] / NVIDIA. – 2009. – Mode of access: http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_BestPracticesGuide_2.3.pdf

Материал поступил в редакцию 11.11.09

UVAROV A.A., SADYKHOV R.Kh. Execution configuration for cuda implementation of image convolution filter

The paper demonstrates approach for constructing optimal execution configuration for CUDA implementation of image convolution filter with different radius. The main goals of execution configuration are to eliminate uncoalesced memory access to improve memory bandwidth and maximize multiprocessor threads utilization. Several mathematical criteria were proposed to choose optimal data block size for execution configuration. Multiple execution configurations based on selected criteria for every filter radius have been formed. It is experimentally find out that two execution configurations with data block size 512x14 and 512x30 are optimal for image convolution filter with radius from 1 to 4. Also described execution configurations are optimal for image processing algorithms which use the same pixel area like considered image convolution filter.

УДК 621.391.826.4

Новиков А.Е., Петровский А.А.

VHDL-РЕАЛИЗАЦИЯ АДАПТИВНОГО КИХ-ФИЛЬТРА НА РАСПРЕДЕЛЕННОЙ АРИФМЕТИКЕ

Введение. КИХ-фильтр генерирует выходной сигнал $y(n)$, равный сумме задержанных входных отсчетов $x(m)$, умноженных на соответствующие коэффициенты:

$$y(m) = \sum_{i=0}^{K-1} w_i x(m-i), \qquad (1)$$

где K – порядок фильтра;

w_i – i-тый коэффициент фильтра.

В обычной реализации фильтр потребует K операций умножения с накоплением (multiply and accumulate - MAC). Для систем, где максимальная тактовая частота ограничена по соображениям энергосбережения, пропускная способность КИХ-фильтра, определяемая как число отфильтрованных отсчетов в секунду, будет сильно ограничена. Это ограничение может стать серьезным при больших порядках фильтров (большое K). Несмотря на то, что применение нескольких процессоров цифровой обработки сигналов (ЦОС процессоров) может повысить пропускную способность, сопутствующее усложнение устройства, увеличение занимаемой площади и потребляемой энергии может сделать такую реализацию устройства малопривлекательной.

Альтернативой может стать реализация MAC операций как серии таблиц с заранее рассчитанными данными, операций доступа к данным и суммирования. Для этого операцию фильтрации (1) нужно представить в последовательном виде. Такая реализация фильтра, известная как распределенная арифметика (РА), позволит поднять пропускную способность за счет более быстрых вычислений, уменьшить сложность устройства ценой повышения требований к объему памяти. Последующие усовершенствования РА позволяют уменьшить объем необходимой памяти, что делает реализацию цифровых КИХ-фильтров на распределенной арифметике более привлекательной.

Одни из первых работ применения распределенной арифметики для задач цифровой обработки сигналов это [1, 2]. Всесторонний обзор цифровых фильтров на РА дан в [3]. В данной работе описана VHDL-реализация адаптивного КИХ-фильтра на распределенной арифметике, который может применяться в системах компенсации электрического эха. Электрическое эхо возникает в гибридной системе при стыковке двухпроводной и четырехпроводной линий связи из-за невозможности согласования импедансов. Результатом является ухудшение качества связи. Наблюдается явление, когда абонент вместе с голосом собеседника слышит собственный задержанный голос. Как следует из стандарта G.168 [4], "существенная часть" импульсного отклика гибридной схемы, которая образует эхо-сигнал, может достигать 0,016 с. При частоте дискретизации 8 кГц, которая используется при обработке сигналов в каналах связи тоновой частоты, длительность такого импульсного отклика будет равна 128 выборкам. Это означает, что адаптивный фильтр должен иметь приблизительно такую же "длину", чтобы компенсировать эхо-сигналы. Согласно [5], при длительности эхо в 0,016 с для обеспечения комфортного разговора уровень эха должен быть ниже уровня голоса минимум на 10 дБ, а при длительности эха в 0,1 с – уже 31 дБ.

Адаптивный КИХ-фильтр. Имеются два абонента – дальний (A) и ближний ($Б$). В случае, если абонент $Б$ говорит одновременно с абонентом A (рис. 1), сигнал от абонента $Б$ равен

$$y_B(m) = x_B(m) + x_A^{echo}(m) \qquad (2)$$

где $x_B(m)$ – сигнал от абонента $Б$;

$x_A^{echo}(m)$ – эхо-сигнал абонента A.

Если говорит только абонент A, то сигнал, приходящий со стороны абонента $Б$, будет равен:

$$u_B(m) = x_A^{echo}(m). \qquad (3)$$

***Новиков Алексей Евгеньевич**, ассистент Белорусского государственного университета информатики и радиоэлектроники.*

***Петровский Александр Александрович,** д.т.н, профессор, зав. кафедрой ЭВС Белорусского государственного университета информатики и радиоэлектроники.*

Беларусь, БГУиР, 220013, г. Минск, ул. П. Бровки, 6.